Cooperative Intelligent Agents for Mission Support

Sanda Mandutianu

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
sanda.mandutianu@jpl.nasa.gov

Abstract—This paper investigates the use of an agent-based framework for knowledge sharing and interoperability between autonomous sub-systems for mission software. The two main goals of this work are the integration of applications into a common framework, and mission independent generalization of applications. The rationale for having multiple interacting agents is that a society of agents would provide added value by cooperation. One of the main benefits is enabling the applications to share information and knowledge and coordinate the decision-making process at run time, in an autonomous manner in the domain of space operations and control. Previous autonomy developments have resulted in efficient and robust inference engines, oriented mostly towards planning and scheduling activities, with little concern for a more general framework of an integrated mission operation system. We attempt to incorporate previous expertise in a more general way, by extending it to a set of heterogeneous applications. The applications will adhere to the common shared background will be generic, mission independent, and re-usable. The system architecture has been based on interacting agents. The applications are considered as knowledge based agents sharing their knowledge. Agents communicate their intentions, goals and they also commit to achieve tasks on behalf of other agents. An agent is more than anything an entity that has goals to fulfill and resources to manage. These goals can be induced by its needs or derived from contracts that the agent commits to achieve. The agents are assigned with goals and they cooperate to achieve these goals. Goals can be specified beyond the level of an isolated agent. Experiments have been realized using a given agent infrastructure, with a representative scenario, requiring the cooperation of different agents; initial results will be presented.

TABLE OF CONTENTS

1. INTRODUCTION

As space missions are continuously increasing in complexity and frequency, the demands upon the flight software systems and ground control personnel are becoming extremely complex. Each of the systems on-board of a spacecraft requires reference to a large volume of diagnostic and correction procedures to cope with the combinatorial process of analyzing possible malfunctions, or events occurring unexpectedly. The mission operations include navigation monitoring, fault diagnosis, scientific experiment management, communication management, etc. It is very important that the software systems designed to support these tasks be flexible, robust, and interactive when possible. Automation of these tasks can improve mission effectiveness, robustness, and lessen dependence on time and bandwidth-consuming remote intervention.

Some of the recent advances in system architecture, mostly oriented towards autonomous control systems, have led to robust execution systems, or interleaved deliberative

and execution systems. In parallel, another trend has been devoted to information systems with a higher degree of autonomy in obtaining and processing the information. Considerably less has been done for a complex application with both such characteristics.

Many of the existing software techniques have not satisfactorily handled the wide variety of problems arising in a space mission. An ideal system should be able to support the decision-making processes either automatically or in an interactive way, depending on the circumstances, to assist with solutions in emergency situations, and to recognize potential problems prior to exceeding alarm limits. It should be also able to integrate and filter different sources of information in solving a particular problem; in doing so, it should communicate with other systems to seek information. During such processes, the system should continuously reevaluate the spacecraft state and redirect the course of action to the most important problems. It is also necessary that the system be able to explain the reasons for its actions.

Achieving this kind of behavior is sensibly beyond the capabilities of conventional systems. It requires the existence of a reasoning mechanism, which based on the system's knowledge about the current situation and about its own abilities can automatically determine a course of action. The system should be able to react appropriately and effectively to new situations in real time, and pro-actively use any available resources to solve the problem, including using the capabilities offered by some other available similar systems.

We present a framework based on interacting agents, i.e. programs that encapsulate specific expertise, solving a problem that might be beyond individual competence. The agent interaction must rely on common and shared concepts and terminology for communicating knowledge across disciplines, and on a communication language. This technology allows the agents to interact at a higher level, expressing concepts related to the application in a general way, independently of the exchange format.

In this paper we discuss the motivations for such an approach and its relevance to space mission systems and we review some initial experiments.

2. AGENT-BASED SOFTWARE ENGINEERING

Agents and agent-based systems represent a new way of thinking, designing and implementing software systems. The agent-based approach brings together a large variety of concepts, models and tools that potentially improve the software engineering of many complex software systems.

It is also important to say that no mention has to be made about the way the system is implemented. It can be realized using different technologies such as AI (Artificial Intelligence), object-oriented models, databases, etc.

Software agent technology covers many different domains and different approaches. Accordingly, a variety of definitions have been produced for the term agent. There are comprehensive discussions on this subject in [Jennings 1998], [Bond 1988], and [Genesereth 1997]. Although at this moment, the definition of an agent still remains fluid, it is considered that there are some characterizing concepts that describe how an agent should act. Conceptually, an agent represents an expressive abstraction for delegating tasks. The agent should act on behalf of its employer. It is generally agreed upon a set of basic characteristics that define an agent. The agent has to be:

- intentional: having a goal or a set of goals (a purpose);
- autonomous: to be able to initiate and determine its course of action without external intervention, once the purpose exists;
- environment aware: possessing a representation of its environment, able to detect changes in that environment and also to make changes to the environment (sometimes this property is called situatedness);
- social: able to communicate with some other agents, humans or artifacts.

It is the presence of all of these characteristics that distinguishes an agent-based system from other systems. There are also some other agent characteristics that could be added to the list, such as adaptability, mobility, but they are not considered indispensable.

In general, autonomous agents perform on behalf of other agents or humans, are able to solve problems on their own, may be geographically distributed, and communicate with each other in order to coordinate their behavior. The aspects of communication and cooperation are fundamental to the definition of agents; these enable a group of agents to solve problems that are beyond the capabilities of any individual agent. This property, of group problem solving, differentiates the software agents from other types of software.

The agent technology offers one of the most powerful and flexible abstractions for *complex software systems*. Such systems can be conceived as multiple communicating agents, which are independently pursuing their tasks or common goals. When interdependencies arise, the agents can communicate with one another and negotiate to handle them. Agents can use legacy software or any non-agent software systems.

The decomposition has been proposed by several other approaches, starting with modular programming, abstract data types, or more recently objects. For some applications, a higher level view, such as the agents as cooperating problem solving entities, represents a more appropriate alternative for the conceptualization process.

3. SPACE MISSION OPERATIONS

There are some domain characteristics that are mostly cited as being representative in choosing agent technology [Bond 1988]:

- data, control, expertise or resources are inherently distributed;
- data system is naturally regarded as an organization of interacting components;
- system contains existing components (legacy systems) which have to be integrated.

A space mission operations system can be considered as a *distributed* software system. We can identify several ways of distributing data, control, expertise or resources:

- distribution of data: the ground mission planning sub-system has different data about the spacecraft than the on-board planning system which has a very specific view about the same spacecraft;
- distribution of control, each sub-system is performing a set of different tasks. The sub-subsystem controlling the camera has different tasks from the navigation system;
- distribution of expertise, the telecom manager knows a lot about specific processes such as link management, which are very different from a science data-mining sub-system;
- distribution of the problem solving entities, the sub-systems are distributed geographically and in the outer space. The spacecraft is capable of sensing some portion of the overall data necessary to solve a problem, while the other part is available on the ground.

For past missions, many of the sub-systems such as the attitude control, navigation, telecommunication management have been considered and operated mostly independently. The expertise on these sub-systems was equally separated into different groups. Members of the design teams might have looked differently at the same problem, using distinct models and sometimes implementing them using different software tools. However, in the design process, the team members shared considerable amount of knowledge, such as the knowledge about the spacecraft building characteristics, the solar environment, and some common mathematical formalism.

As new, more complex and demanding missions are considered, the different disciplines become more interdependent. Such examples are flight through a planetary atmosphere, such as landing or ascending from a planet or the Earth, and spacecraft formation flying where the relative configuration and operation must be closely controlled. The commonality between different sub-systems has to be represented in the design of a unified system that can be easily reconfigured to serve the needs of a wide set of future missions.

Although in an integrated system one might still consider the possibility of unifying the models, representations and tools, the problem can be naturally modeled as a set of autonomous problem solvers, or agents, with their own resources and expertise. In order to solve a problem requiring all their combined distributed capabilities, they have to *interact* and possibly share some common knowledge to get the task

done, while maintaining their own expertise and ways of exercising it.

A simple example would be the task to take a picture and transmit it to the ground. A great part of the problem can be carried out independently by distinct agents, doing their job independently of one another and in parallel, with a few information exchanges at a very high level. Examples of exchanges might include a request made by the navigation agent to the camera agent to take a picture, or a request made to the telecom manager to transmit the picture to the ground after the picture was taken.

Any of the sub-systems of a space mission might justifiably address its problems in very particular ways, using different models, making use of particular representations and reasoning methods to do their work efficiently. Furthermore, achieving a task is often a process of *negotiation*; decisions are made and changed based on the current internal state of the agents and the state of the environment. For instance, an agent might decide in favor of one of the available ground stations to downlink data for a particular satellite, depending on local factors as weather, or might decide to skip one step in a procedure due to a failure of a particular equipment on the spacecraft, or change the course of action due to a change in the way of thinking about a particular situation. These decisions, currently made by a human factor, can be transferred to an autonomous procedure having the necessary knowledge to make a safe decision.

Developing software systems for space missions lead to the existence of numerous *legacy systems*, very well designed and developed to solve a particular problem. There is a challenge for an integrating system to use as non-intrusively as possible such very useful software artifacts. Sometimes the diversity was due to a design decision, or was imposed by the software development tool that was employed. There was often the case that the application itself required an efficient approach, even if the solutions work reasonably only in a particular, restricted domain. Such representational differences between systems constitute serious challenges when they have to be integrated in a more complex application. This ends up in lack of consistency between different models and therefore prevents integration and interoperability between the constituent applications. The diversity should be taken as a given; there will always be preferences towards particular tool suites and integration environments, and there will be always a better solution for a given problem. Nonetheless, for systems that involve large and different segments of a project, or multiple projects, the processes that take place have to be coordinated.

4. AGENT CONTROL

In this paper an agent is considered as an entity described in terms of common sense modalities such as beliefs, or intentions. This intuitive approach has its benefits for complex systems such as distributed systems, robots, etc. where simpler, mechanistic descriptions might not exist. It has to be mentioned that what makes agents out of hardware or software components is the fact that they are analyzed and controlled using these common sense, mental terms.

Intuitively, each agent represents an independent problem solver. Accordingly, an agent has a general problem solving mechanism and more specialized problem oriented capabilities. The agent has also to possess communication capabilities that make it able to communicate adequately with other agents. Agents are deliberative, in the sense that they use their knowledge in support of their reasoning.

The agent structure we use has been conceptually inspired by Shoham's agent-based programming approach [Shoham 1995], with some additions from the BDI (Beliefs, Desires, Intentions) model [Rao 1995], although the practical realization might differ in some aspects. The main idea is an analogy to those conceptual categories proved useful in designing the agent control architecture. It is assumed that the agent has to accomplish complex tasks that require higher level abstractions extracted from its behavior in relation with its environment, and *with other agents*.

An agent-based framework includes the agent state, and an interpreted programming language used to define agents. The agent state is defined by four basic components:

- *beliefs*: the information that the agent has about the environment and about some other agents;
- *capabilities*: the tasks that are supposed to be accomplished by the agent under given circumstances;

- *commitments*: the tasks that the agent is committed to achieve (usually communicated to another agent) at a particular time;
- *intentions*: the decisions about how to act in order to fulfill its commitments . This concept is necessary because it is assumed that the agent wouldn't be able in general to

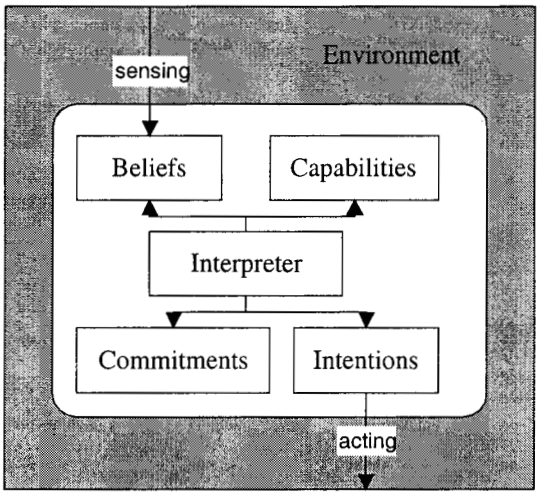


Fig. 1 Agent State

achieve all its commitments (desires).

It is assumed that this set of concepts are necessary to model agent behavior such as: deliberation, reactivity, interaction, and are flexible enough to be used at different levels of the agent architecture to describe the agent state.

To be realized, the above mentioned concepts have to be represented by data structures in the agent architecture. Besides what was already mentioned, to complete the architecture, the agent might also have a repository of recipes (plans or rules) which specify the courses of actions that may be followed by the agent to achieve its intentions.

The beliefs are updated from observations made of the world and as the effect of the interactions with other agents, generating new capabilities and commitments on the basis of new beliefs, and selecting from the set of currently active commitments some subset to act as intentions (Fig. 1). An action has to be selected based on then agent's current intentions and knowledge (beliefs plus plans/rules).

The agent behavior can be expressed in a declarative *agent language*, which can be used by an interpreter that controls the agent execution cycle.

The agent execution cycle consists of the following steps: processing new messages, determining which rules are applicable to the current situation, executing the actions specified by these rules, updating the mental model in accordance with these rules, and planning new actions.

Although not yet fully theoretically defined and still lacking clear definitions, these systems appear to have significant advantages in complex applications characterized by uncertainty, uncontrolled environmental change, and ongoing changes in the specification. One of the key advantages is that they offer a level of abstraction much closer to the specification of the real problem, where it is possible to make statements like "Agent A wrongly believes that the antenna B is available for transmission".

In order to achieve *coordination* in a multi-agent system, the agents might have to negotiate, and they have to exchange information, i.e. they need to communicate. In multi-agent systems, the possible solutions to the communication range from no communication at all, ad hoc agent communication languages, and standard agent Communication languages. If the agent communication language (ACL) is primitive, requests, commands and complex intentions cannot be expressed. If an ad hoc language is used, this makes the inter-operation non-trivial, and sometimes impossible.

A more comprehensive description of the agent control process is shown in Fig. 2.

We've started by using an existent agent communication language- KQML (Knowledge Query and Manipulation Language) [Finin 1992] although the maturity of the agent languages has yet to be proven. We prefer this approach instead of ad hoc communication solution like message passing, or using a blackboard (in fact no communication language at all) which make the issue of interoperation between different applications very difficult to solve.

KQML is both a language and a protocol. It can be viewed as being comprised of three layers: a content layer, a message layer and a

communication layer. The content layer is the actual content of the message, in a particular representation language. KQML can carry any representation language, including logic languages, ASCII strings, etc. All of the KQML implementations ignore the content portion of the message except to the extent that they need to determine the message limits.

The message and communication layers encode features describing the lower level communication parameters, such as the identity of the sender and the recipient, a unique identifier associated with the communication process, and the kinds of interactions one can

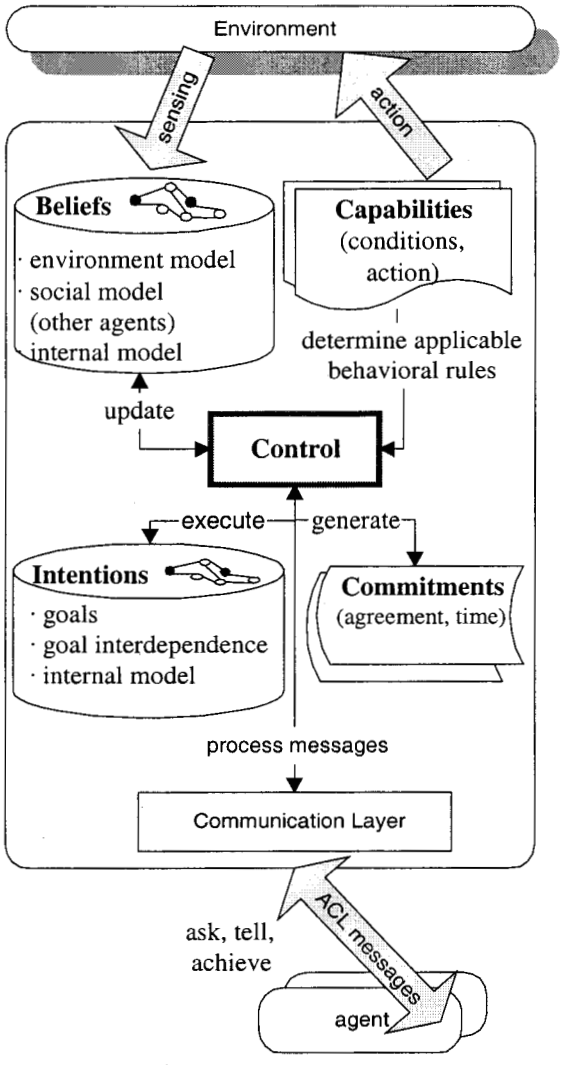


Fig. 2 Agent Functional Structure

have with the KQML-speaking agent. This way, the sender can attach a type of the communicative interaction, known as a performative, such as an assertion, a query, or a command (tell, ask, achieve). A KQML message conceptually consists of a performative, associated arguments including the real content of the message and a set of optional arguments describing the content in a manner that is independent of the content language. For example, a query about the availability of a particular antenna might be represented as:

(ask-all :content (AVAILABLE (?antenna))
:ontology MISSION-MODEL)

5. SYSTEM ARCHITECTURE

The system architecture presented here is based on interacting agents. The applications are considered as knowledge based agents sharing their knowledge by using an agent communication language. Agents have to communicate their intentions, goals and also they will commit to achieve tasks on behalf of other agents. An agent is more than anything an entity that has goals to fulfill and manages resources. These goals can be induced by its needs or derived from contracts that the agent commits to achieve. The agents are assigned with goals and they cooperate to achieve these goals. Goals can be specified beyond the level of an isolated agent.

The infrastructure for such an open knowledge sharing system can contain different knowledge representation structures, intercommunication format(s), one or more knowledge manipulation languages, a common shared model or set of models (ontology) and a software framework to allow for the development of actual software systems.

For example, there are shared elements of knowledge between the diagnoses of anomalies occurring on-board the spacecraft and the processing required to generate the necessary operational procedures needed to recover from such anomalies, based on the spacecraft design model. In order to control the run time (autonomous) re-planning process, certain knowledge needs to be shared dynamically. Identifying this type of knowledge is the first step to defining ontology to support the integration of a set of generic mission support and mission operations tools that cooperate to

solve operations tasks; ontology represents the application domain.

The open knowledge based software architecture can be described by:

- the representation structure;
- the knowledge communication format;
- the knowledge communication and manipulation language;
- a common shared model of the domain (ontology).

The *representation structure* is provided by the concept of virtual knowledge base. The applications can preserve their specific representations, while maintaining a virtual common representation structure. The virtual knowledge base represents a declarative representation of the agent's beliefs (what it knows about the environment), capabilities and goals.

The *knowledge communication format*, as a "content language" might be a formally defined generic language, or some private languages might be still considered for efficiency reasons. A "content language" is used to express declarative knowledge within "performatives" (communication primitive acts such as ask, tell, etc.) provided by protocols by which knowledge bases, encoded in whatever representation can be accessed. The content language might be as simple as specification of common libraries of objects/classes.

To support sharing of knowledge at execution time as opposed to the off-line incorporation of reusable modules, there need to be protocols to control the exchange of knowledge according to the intentions of various interacting agents. This is the purpose of a *knowledge communication and manipulation language*. One can make a distinction between the "content" of a communication (e.g. "antennas are available") and the "intention" of a communication. The same content can be communicated with different intentions. For instance, a question is communicated as "which antennas are available?" and a retraction as "it is no longer true that the antenna X is available". The core set of performatives defines a basis upon which higher levels of communication can be expressed, such as negotiation, cooperation, etc.

It is assumed that the agents communicate at a higher level of discourse, the contents of the communication being meaningful statements about the environment or about other agents. This differentiates agent communication from other computational entities such as method invocation in CORBA.

The *ontology* proves to be essential, providing the applications with a common conceptualization of the domain, in our case spacecraft mission operations and control. Using a common description of the domain (the ontology) creates for the application a virtual shared model without the constraints that occur in centralized or uniform architectures.

For example, a planning system has been based on a theory (model) in which plans are composed of "activities" which form "goals" with specific "resource dependencies" and the search for plans has been guided by "ordering heuristics" and "optimization criteria". If some other applications want to use this planning system, it is necessary to have a common understanding of what these concepts mean, preferably organized into a knowledge base with specific knowledge.

Ontology is a concrete means to achieve knowledge sharing. Operationally, an ontology (a model of the domain) consists of a set of definitions of the representational terms used in the definition of the knowledge bases. The ontology implementation might vary from a re-usable shared library, to formally defining the ontology elements such as the knowledge representation models, or a semantic network.

6. AN INTELLIGENT AGENT MISSION EXPERIMENT

To demonstrate the suitability of agent-based architecture for mission operations, we started with a simple but representative scenario where concepts such as model sharing, ontologies and interoperability will be adequately represented.

Let's consider the telecom management modeling as an example. A telecommunication and radio navigation network, the Deep Space Network (DSN) supports the communication between the spacecraft and the ground stations. DSN receives telemetry signals from the spacecraft, sends commands to control spacecraft operations, generates the radio navigation data used to locate and guide the spacecraft, etc. (Fig.

3). The telecommunication link between a DSN node and the spacecraft provides navigation data as well as various other communication data. The telecommunication links might be established using different areas of the electromagnetic spectrum; the communication system has to support all the specific data types. There might be commonality in describing the data, or using the same methods of analyzing the data. The common aspects need to be abstracted, so different agents can share the same knowledge.

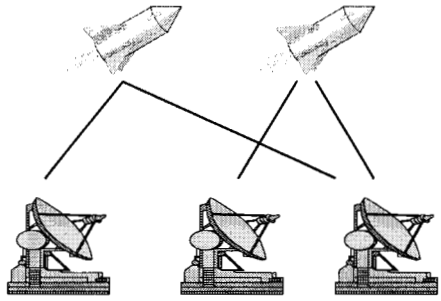
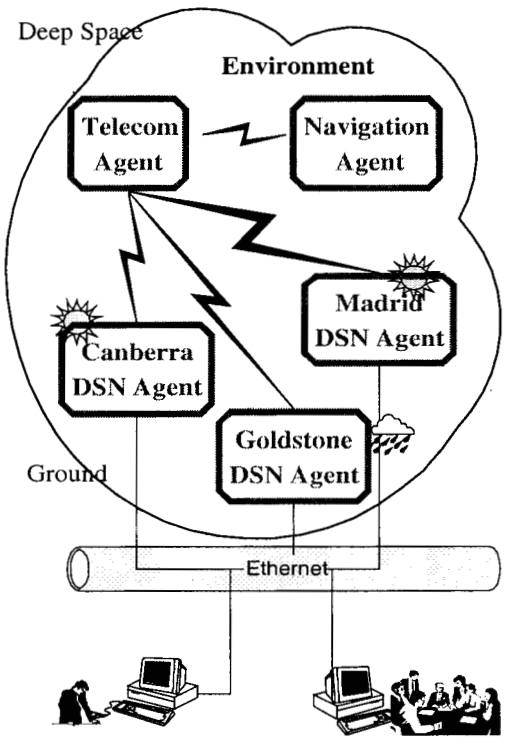


Fig. 3 A Telecommunication Scenario

A simple scenario would be the transmission of mission goals from ground to the navigation agent. The navigation agent defers the telecommunication goals to a telecommunication agent. At its turn, the telecom agent has to decide what is the best choice from the available ground stations.



An agent represents each ground station. Typically, the planning and scheduling have been coordinated for both partners- DSN and spacecraft, on the ground. Generalizing, any partner might have the ability to request data. The partner must be notified and must cooperate to configure the link properly for the requested data flow. The scheduling process has to take into account constraints and requirements that are not the same as those for the telecommunication processes only. There will be probably constraints of this sort associated with the spacecraft-to-spacecraft links or other non-Earth links.

This apparently simple scenario involves such issues as:

- distribution of decision making processes between spacecraft, and ground complexes;
- interaction between applications developed with different purposes, that have to share knowledge to solve a particular problem;
- local telecommunication performance analysis, at both ends (spacecraft and ground);
- initiate activities by either participants;
- negotiate solutions between agents;
- identification of common knowledge to be shared by agents.

The following agents have been identified as typical for the given scenario (Fig.4):

- **Navigation Agent**: ensures that the spacecraft maintains its trajectory, controls and monitors other on-board activities such as capturing images, achieving science experiments, etc. It receives a set of goals from the ground, and attempts to achieve them or transfer them to other agents;

- **Telecom Agent**: controls the on-board telecommunication processes between the spacecraft and the ground stations. It communicates also with the navigation agent to coordinate the downlink related activities;
- **Ground Complex System Agents**: controls the telecommunication processes on the ground. There are as many agents as complex systems on the ground.

The agents, which might be heterogeneous, share some of their knowledge. They do this by interacting at run-time, based on a common set of models, grouped in a shared ontology. The substrate of this process is the concept of virtual knowledge base every agent has (Fig. 5).

communication channel established between the spacecraft and the ground). Ontology can be represented as a semantic network as shown in Fig. 6, and implemented as a library of shared objects.

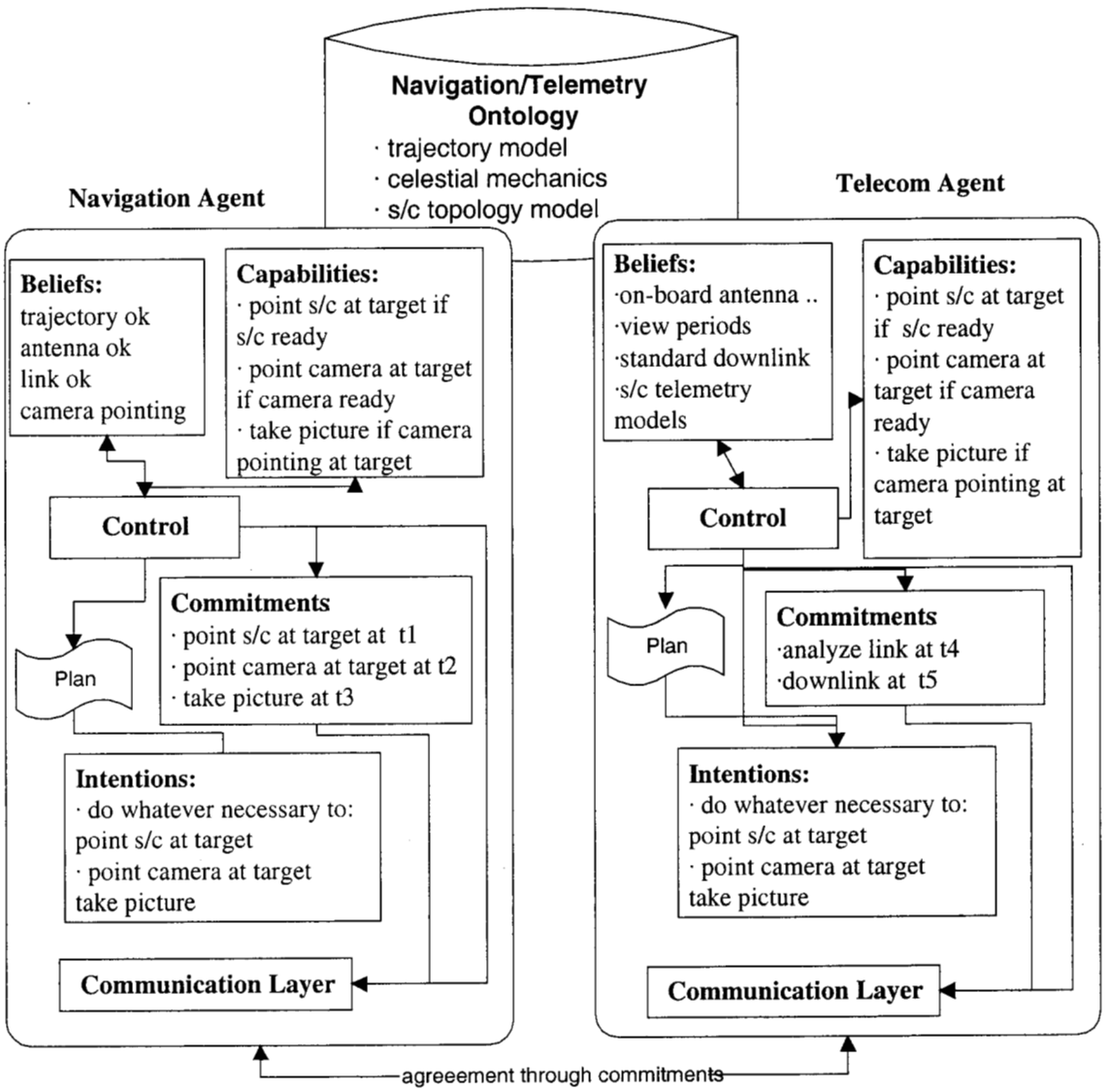


Fig. 5 Navigation/Telecom Cooperating Agents

Some possible elements in a navigation/telecom ontology include: antenna, ground complex, navigation, and telecom link (the instant

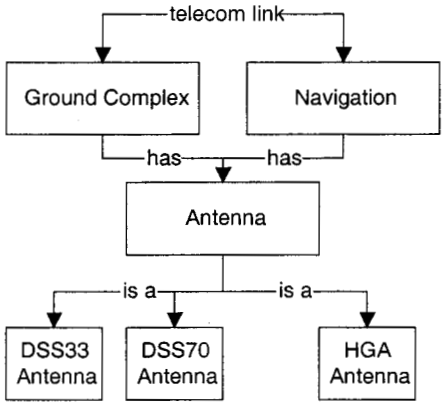


Fig.6 A semantic network for a navigation/telecom ontology

The agents can reason about what to do based on their current state represented by beliefs, capabilities, existing commitments, etc.

Beliefs represent the current state of the agent. Some examples of beliefs, given in pseudo-code as a pair of belief name and belief value type, are given in Fig.7.

Position	(Coord1, Coord2)
Goal	(String, StartTime, EndTime, Duration, Status)
Velocity	(Float)
StartTime	(Time)
EndTime	(Time)
Duration	(Integer)
Coord1	(Float)

Fig. 7 Beliefs Example

The beliefs have to be expressed using RADL, or AgentBuilder current conventions. Essential is that one can explicitly declare and refer agent's beliefs.

The agent has to perform actions, based on action's necessary preconditions. The agent capabilities define the actions that the agent can perform provided their preconditions are satisfied. Capabilities, as opposed to beliefs, are given for the life of an agent. Beliefs can change, by adding new beliefs or updating existing ones. The actions can be either effect the environment in some way, or may be accomplish a communicative act.

An agent can perform physical actions as pointing the spacecraft to a target, turning the camera on, achieve a science experiment, send data to the ground (downlink), etc. It also can send a message to some other agents asking for some information, may receive messages requesting goal achievement, and so on. Capabilities are pairs of action and preconditions to be fulfilled for that action to take place.

For example, consider the telecom Agent has to downlink data. Some of the actions of this agent are analyze the telecom link, choose a ground complex to downlink, downlink. A capability for

Action:	downlink
Preconditions:	on-board antenna ok, ground complex available, data rate satisfactory.

Fig. 8 A Capability Example

analyzing the telecom link is given in Fig.8:

Capabilities are used by the agent to decide to commit to achieve a particular action. A commitment is an agreement, usually communicated to some other agent to perform a particular action at a certain time (Fig. 9).

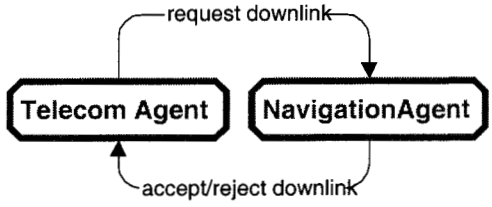


Fig.9 Inter-agent Communication

For instance, if the ground complex accepts the request to downlink, it agrees to perform the requested action at the requested time, based on the details of the request, its behavioral rules and its current mental model (beliefs, existing commitments, etc.). It is to be noted that a

commitment does not guarantee that an action will be performed; more precisely, a commitment is an agreement to attempt a particular action at a particular time if the necessary preconditions for that action are satisfied at that time.

In general, successful execution of an action may be beyond the agent's control. For example, a Ground Complex Agent has committed to accept the downlink on behalf of a Telecom Agent. Even if the necessary preconditions are met and Telecom Agent is able to initiate execution, the action may still fail (e.g. a crash during the transmission). The Ground Complex Agent must monitor the execution so it will be able to send a message back to the Telecom Agent to report the success or failure of the commitment.

The communicative actions such as asking for information, making something true about the environment, passing information to another agent, passing information to all agents are expressed using KQML performatives: ask, achieve, tell, broadcasts, etc. A KQML message consists of a performative, the content of the message and a set of optional arguments. The performative refers to the contents of the virtual knowledge base of a remote agent.

The arguments describe the content of the message independently of the content language. For instance, a message representing a query asking about the availability of a ground system complex might be represented as:

```
(ask-all :content (downlink (?datarate)(?volume))
         :language any-language
         :ontology navigation-1)
```

In this message, the performative is ask-all, the content is (downlink (?datarate)(?volume)) and the ontology is identified by "navigation-1". The content might be expressed in any language that is understood buy the communicating agents.

The inter-agent communication and interoperability is based on KQML. The agents might be running on different platforms and might be using different content languages for communication.

The agent behavior is described by its behavioral rules. The behavioral rule extends the commitment rule by determining the course of action that the agent takes throughout execution.

```
("Forwarding downlink selection"

WHEN
IF
(BIND [VAR Goal <?g>])
(OBJ [INST Goal <?g>.name] EQUALS
     [VAL String "image of Europa on
ground"])
(OBJ [INST Goal <?g>.status] EQUALS [VAL
String "accepted"])
(OBJ [VAR Agent <?agent>.agentInfo.name]
EQUALS
     [VAL String "Telecom"])
THEN
(DO SendKqmlMessage ([NEW KqmlMessage],
                [VAR
Agent<?agent>.agentInfo.name],
                [VAL String "achieve"],
                [VAR Goal <?g>],
                [],
                [], [], [], [], []))
)
```

Fig. 10 Behavioral rule- forwarding a goal

An example of a behavioral rule is given in Fig. 10. The rule is written using RADL (Reticular Agent Description Language), a proprietary agent language [AgentBuider 1998].

RADL has extended the idea of a commitment rule to a general behavioral rule. Behavioral rules match the set of possible responses against the current environment described by the agent's beliefs and against the messages received from other agents.

Behavioral rules can be seen as production rules where the IF part matches against beliefs, commitments and intentions, and the WHEN portion matches against messages. The THEN portion represents agent's actions and belief changes performed in response to the current event, internal beliefs and external environment.

The rule in Fig.10 states that if the goal "image of Europa on ground" exists as a belief, and it was accepted as feasible (goal status is "accepted"), then it will be forwarded to the telecom agent within an "achieve" message.

In this architecture, the agent interpreter continually monitors incoming messages, updates the agent's internal model and performs or initiates appropriate actions. The internal model contains current beliefs, commitments, intentions, capabilities and rules of the agent. Although rules and capabilities are static, the agent's beliefs, commitments and intentions are dynamic and can change over the agent's lifetime.

7. FINAL REMARKS AND FUTURE WORK

The two main goals of this work are the integration of applications into a common mission framework, and mission independent generalization of applications. There are also several other ways that the agent-based software engineering might prove to be beneficial to mission operations:

- increased flexibility, by providing advice and making independent decisions;
- increased robustness by distributing decision making processes;
- increased system efficiency by achieving tasks in parallel;
- integration by coordination, and not by common representation;
- interoperability across heterogeneous languages and platforms;
- integration of legacy systems.

There have been several agent-based approaches to system control and in particular to spacecraft control or mission operations. We can cite the Remote Agent Experiment for automatically spacecraft controlling and commanding NASA DS1 mission [Bernard 1998], or multi-agent approaches such as multi-operation support operations [Siewert 1996], and handling malfunctions in the Reaction Control System (RCS) of NASA's space shuttle [Ingrand 1996]. Some of them solve the problem using one single agent, such as for DS1, or in the case of a multi-agent approach have no explicit model for inter-agent communication or application integration. There is some other multi-agent approaches with more elaborate models for cooperation between agents such as in [Tambe 1995].

We hope that the experiments with a given infrastructure for building multi-agent systems, will help to better understand fundamental issues such as the nature and content of a mission models, how to use them and how to express them in such a way that they might be re-used. By conceptually considering the mission system as a distributed process, the agent-based engineering certainly helps to better define the components of an application and offers support for an advanced problem solving entities framework.

Further work should focus on defining mission ontology, refining the communicative models to allow for complex interactions such as negotiation.

8. REFERENCES

1. [AgentBuilder 1998]: User's Guide, Reticular Systems, Inc., 1998.
2. [Bernard 1998] Bernard, D.E., et al.: Design of the Remote Agent Experiment for Spacecraft Autonomy, Proc. of the IEEE Aerospace Conference, Aspen, 1998.
3. [Bond 1988]: Bond, A., H., Gasser, L., eds.: Readings in Distributed Artificial Intelligence, Morgan Kaufmann Publishers, San Mateo, CA 1998.
4. [Finin 1992] Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., McGuire, J., McKay, D., Shapiro, S., Pelavin, R., Beck, C.: Specification of the KQML Agent Communication Language (Official Document of the DARPA Knowledge Sharing Initiative's External Interfaces Working Group), Technical Report 92-04, Enterprise Integration Technologies, Inc., Menlo Park, California, 1992.
5. [Genesereth 1997] Genesereth, M., R.: An Agent-Based Framework for Interoperability. In Software Agents, J.M. Bradshaw, AAAI Press/MIT Press, 1997.
6. [Ingrand 1996] Ingrand, F.F., Georgeff, M.P., Rao, A.: An Architecture for Real-Time Reasoning and System Control, http://www.laas.fr/~felix/publis/ieee-exp92/ieee-diagl2h.html
7. [Jennings 1998] Jennings, N.R., Sycara, K., Woolridge, M.: A Roadmap of Agent Research and Development. In Autonomous Agents and Multi-Agent Systems, 1, 275-306 , Kluwer Academic Publishers, Boston, 1998.

8. [Rao 1995] Rao, A., Georgeff, M.: BDI Agents: From Theory to Practice. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, June 1995.
9. [Siewert 1996] Siewert, S.: A Distributed Operations Automation Testbed to Evaluate System Support for Autonomy and Operator Interaction Protocols. In Proc. of 4th International Symposium on Space Mission Operations and Ground Data Systems, ESA, Forum der Technik, Munich, Germany, September 1996.(http://www-sgc.colorado.edu/people/siewerts/intl_space_ops_96/SO96_6_07.html
10. [Shoham 1995] Shoham, Y.: CSLI Agent-oriented Programming Project: Applying software agents to software communication, integration and HCI (CSLI home page), Stanford University, Center for the Study of Language and Information, 1995.
11. [Tambe 1995] Tambe, M., Johnson, W.,L., Jones, R.,M., Ross, F., Laird, J., E., Rosenbloom, P.,S., Schwamb, K.: Intelligent Agents for Interactive Simulation Environments, AI Magazine, Spring 1995.

ACKNOWLEDGEMENTS

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, and it was sponsored by the JPL Director's Research and Development Fund and by the JPL Deep Space Network Technology Development Plan. Thanks to Peter Shames, Kar-Ming Cheung, Abdullah Aljabri, Pierre Maldague, Adans Ko, James Renfrow and Robin Vaughan for their support and valuable discussions.

Sanda Mandutianu is a senior member of the technical staff at Jet Propulsion Laboratory, California. She holds an MS in Physics from University of Bucharest, Romania. She has been working and leading projects in Artificial Intelligence since 1983 in various fields such as knowledge representation, cognitive science, natural language understanding, distributed computing and applications. Her current interests are cooperative software agents, agent communication languages and spacecraft autonomy.